OpusFilter : A Configurable Parallel Corpus Filtering Toolbox

Aulamo, Mikko

The Association for Computational Linguistics
2020-06-19

Aulamo , M , Virpioja , S & Tiedemann , J 2020 , OpusFilter : A Configurable Parallel Corpus Filtering Toolbox . in A Çelikyilmaz & T-H Wen (eds) , 58TH ANNUAL MEETING OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS (ACL 2020): SYSTEM DEMONSTRATIONS : System Demonstrations . The Association for Computational Linguistics , Stroudsburg, PA , pp. 150-156 , 2020 Annual Conference of the Association for Computational Linguistics , Seattle , Washington , United States , 05/07/2020 . https://doi.org/10.18653/v1/2020.acl-d

http://hdl.handle.net/10138/319556
https://doi.org/10.18653/v1/2020.acl-demos.20

OpusFilter: A Configurable Parallel Corpus Filtering Toolbox

Mikko Aulamo and **Sami Virpioja** and **Jörg Tiedemann**
Department of Digital Humanities
University of Helsinki, Helsinki / Finland
{mikko.aulamo, sami.virpioja, jorg.tiedemann}@helsinki.fi

Abstract

This paper introduces OpusFilter, a flexible and modular toolbox for filtering parallel corpora. It implements a number of components based on heuristic filters, language identification libraries, character-based language models, and word alignment tools, and it can easily be extended with custom filters. Bitext segments can be ranked according to their quality or domain match using single features or a logistic regression model that can be trained without manually labeled training data. We demonstrate the effectiveness of OpusFilter on the example of a Finnish-English news translation task based on noisy web-crawled training data. Applying our tool leads to improved translation quality while significantly reducing the size of the training data, also clearly outperforming an alternative ranking given in the crawled data set. Furthermore, we show the ability of OpusFilter to perform data selection for domain adaptation.

1 Introduction

Data filtering tools are important to reduce the noise fed into machine learning algorithms such as the ones used in neural machine translation. This is especially true for data sets with suspicious sources like unrestricted web crawls or data sets that are automatically extracted from complex data formats such as PDF or HTML with all their different flavors and implementations. Cleaning parallel corpora is a special case in which not only the raw data but also the quality of alignment between source and target language needs to checked. The aligned translations drive the mapping from input to the output language as a strong supervision during the training steps, and the amount of noise will have a decisive impact on the adequacy of the translations. The effect is especially severe for low resource settings, in which little data is available, and each mistake might directly influence the end result.

The interest in automatic bitext (i.e. bilingual parallel corpora) filtering is constantly growing pushed by the advances in neural machine translation. Khayrallah and Koehn (2018) show that noisy training data is often more harmful for neural translation models than statistical translation models. As a consequence, international evaluation campaigns like the ones organised by WMT now feature shared tasks on data cleaning and ranking (Koehn et al., 2018, 2019). Various approaches have been proposed based on such challenges and directly benefit the development of MT engines in low-resource settings.

This paper presents a framework for bitext cleaning, OpusFilter, focusing on processing data collected in OPUS (Tiedemann, 2012), the world's largest resource of openly available parallel corpora. In contrast to tools such as bicleaner (Sánchez-Cartagena et al., 2018) and Zipporah (Xu and Koehn, 2017), that implement a single method for parallel corpus filtering, OpusFilter is designed as a toolbox that is useful for testing and using many different approaches. Below we describe the design of OpusFilter and present its application in the test case of filtering Finnish-English parallel data included in ParaCrawl.

2 OpusFilter Toolbox

The OpusFilter toolbox is implemented in Python 3 and is available at `https://github.com/Helsinki-NLP/OpusFilter` under the permissive MIT open-source license. The main script provided by the package is `opusfilter`, which takes a configuration file as an input. The configuration files are written in YAML syntax.[1] A configuration contains common global options (currently only the output directory) and a list of steps that are run one by one. There are different step types

[1] See https://yaml.org/

Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 150–156

(functions) for downloading parallel corpora from the OPUS database, combining and taking subsets of the corpora, filtering and scoring segment pairs with a combination of different filters, and training and using a classifier based on the scores. The configuration files for all our experiments are included in the GitHub repository.

The input and output files for the functions are defined in the configuration, and it is simple to also use external data files. In contrast to common text file processing tools, the OpusFilter functions support the processing of parallel files that have corresponding data on the same lines. Special attention has been paid to make the processing of large files memory-efficient: Full corpora are never loaded into memory, but the segment pairs and scores are processed one at a time if possible, and in fixed-size chunks otherwise.

In this section, we describe the current functionality of the OpusFilter toolbox. In the future, we plan to include more functions for common monolingual and parallel data processing operations. The ultimate goal is that all pre-processing steps could be defined in a single configuration file making it easy to share them for reproducing MT experiments.

2.1 Downloading and selecting data

The first steps typically refer to data selection and their preparation. Relevant data sets can be downloaded, concatenated, and divided into subsets.

opus_read uses the OpusTools[2] library to download a specified parallel corpus from the OPUS corpus collection, and stores it into two files (source and target segments, one segment per line). There are options for selecting a specified version of the corpus and whether to download pre-tokenized or untokenized segments.

Multiple files can be concatenated by the **concatenate** function. The **head**, **tail**, and **slice** functions can be used to the lines from the top, bottom or middle of the parallel input files. Furthermore, **subset** takes a random subset of the selected size from a corpus. It has an option to shuffle the target language segments to produce examples of poor translation pairs that can be used as negative examples in training a segment pair classifier. Finally, **split** divides parallel files into two parts when given the approximate proportions as fractions. The split is based on a hash function, making it deterministic on the content of the input lines.

2.2 Filtering and Scoring

All filter classes implemented in OpusFilter are applicable both for direct filtering of the data and for producing a quality score for each segment pair. If a filtering method does not produce any sensible score, it should output 1 for acceptable pairs and 0 for unacceptable pairs. Any method that produces one or more scores provides options for selecting filtering thresholds for the scores.

The current filters implemented in OpusFilter include (a) simple length-based filters (maximum and minimum length and segment length ratio in words or characters), (b) script and language identification filters, (c) filters that consider special characters such as numbers and punctuation marks, (d) filters that use probabilities from n-gram language models, and (e) filters that use word alignment probabilities. For a complete list, see the documentation of the software.

The filters can be used by two functions:

filter applies a specified list of filters to a parallel corpus and outputs those segments that pass all the filters (or optionally those that do not).

score produces scores for the segment pairs in a parallel corpus from the specified list of filters. The scores are written in JSON Lines format, which is easy to process, and for example simple to load as a `pandas`[3] `DataFrame` object.

There are also methods for using and processing the score files: **join** is a function to combine separate score files to a single file, and **sort** sorts the given input files based on the scores. Reordering the data makes it convenient to remove noisy pairs from the end of the sentence files.

In addition, OpusFilter implements **remove_duplicates** for filtering out duplicate lines from parallel corpora. The matching can be based on any combination of the lines in the input files, so that it is possible, for example, to make sure that each target sentence occurs only once in a bitext.

2.3 Classification

The scores calculated by different filters can be used as features for a classifier that predicts whether a given segment pair is clean enough to be used, for example, for training machine translation models. Moreover, the classification probability

[2] https://github.com/Helsinki-NLP/OpusTools

[3] https://pandas.pydata.org/

can be applied for sorting the data according to their expected cleanliness.

The classification approach currently supported by OpusFilter is inspired by Vázquez et al. (2019). First, we take a set of sentence pairs and score them using features produced by filters. This set is then split into clean and noisy examples in order to be used as the training data for a logistic regression classifier. To choose the positive and negative example pairs, we set a percentage threshold value for all filter scores. Each sentence pair has to obtain scores that are above the threshold percentile for all filters in order to be considered clean; otherwise, they are labeled noisy.

Unlike Vázquez et al. (2019), who manually placed the threshold between the two peaks of a score distribution in cases where the distribution is bimodal, we implemented an automatic selection of the optimal threshold to ensure a more convenient usage of the OpusFilter toolbox. Multiple models with different training data splits using different thresholds can be trained in order to find the best performing model. The minimum, maximum, and initial percentage thresholds can be specified for each score in the configuration file, and optimized with a search algorithm. The optimization criterion can be cross-entropy of the classifier[4] or the area under the receiver operating characteristics curve (ROC AUC) based on a development set of scores labeled as noisy or clean by the user.

Finally, once the logistic regression model is trained and selected, it can be applied to each segment pair in a larger set of data to produce a single cleanness score, which is the probability prediction from the model. For classification, the following functions have been implemented:

train_classifier optimizes a classifier to predict the cleanliness of the segment pairs using the procedure described above. The inputs are training scores, the criterion to be used in the model optimization, search algorithm details for the optimization, and a development set if the ROC AUC criterion is used. The optimized classifier is written to the specified output file.

classify assigns either a cleanness score or label to each sentence in a data set. The inputs are the classifier file and the sentence pairs to be classified, and the resulting scores or labels are written line by line into a specified output file.

2.4 Custom Filters

The toolbox is extendable with custom filter classes defined in Python. The filter classes should be based on the abstract base class `FilterABC` and implement two methods: `score` and `accept`. The `score` method takes an iterator over segment pairs, and yields a score object for each pair. The score may either be a single number, or if multiple score values need to be yielded, a dictionary that has the numbers as values. The `accept` method takes a single output yielded by the `score` method, and returns whether the segment pair should be accepted based on the score.

2.5 Studying Filter Scores

In addition to the main `opusfilter` script, there is a separate tool `opusfilter-scores` for calculating and plotting statistics from scored segment pairs. The commands include `describe` for printing the basic statistics of the scores, `hist` for plotting score histograms (see the example in Figure 1a), `corr` for plotting a correlation matrix of the scores (Figure 1b), and `scatter-matrix` for drawing a matrix of scatter plots between the values of different scores.

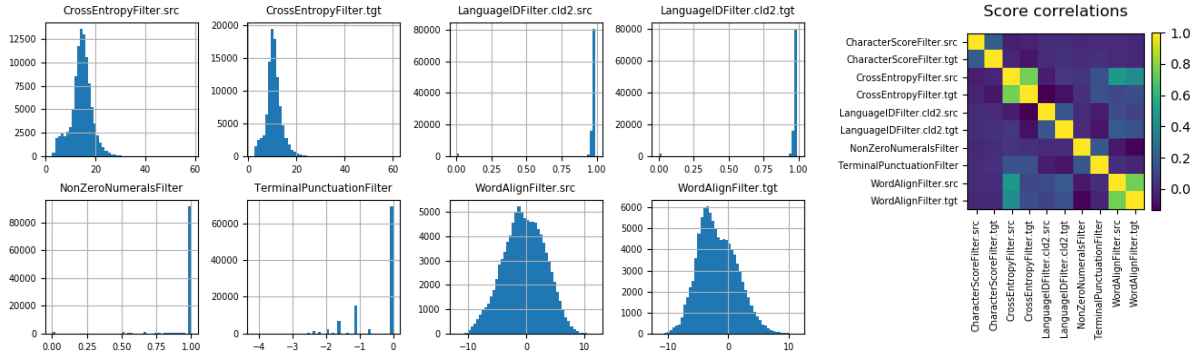
3 Experiments

To demonstrate the usefulness of the OpusFilter toolbox, we show results from two main experiments on the Finnish-English news translation task (in both directions): (i) Filtering noisy data, and (ii) applying domain adaptation.

For training, we use data from version 4 of the ParaCrawl corpus (Esplà-Gomis et al., 2019). The data is taken from a general internet crawl and contains segments that are noisy and potentially harmful for machine translation models. We use the subset of the corpus that is already filtered by the bicleaner tool[5] (Sánchez-Cartagena et al., 2018). This data set contains 2,156,069 segment pairs and is ordered by the score from bicleaner, which enables us to directly compare it to our tool. We create five versions of the training data by removing 10%, 20%, 30%, 40% and 50% of the pairs from the noisy end of the collection and train translation models with the full data and with the five reduced

[4] Also, Akaike Information Criterion (AIC) or Bayes Information Criterion (BIC) can be applied, similarly to how Vázquez et al. (2019) operate in cases where the score distribution is not bimodal. However, they differ from the cross-entropy only in the case that a feature can be completely removed.

[5] https://github.com/bitextor/bicleaner



(a) Histograms of the scores.

(b) Correlations between the scores.

Figure 1: Histograms and correlations of the score values used for training classifiers in the Finnish-English noise filtering. CharacterScoreFilters have been excluded from histograms as their values are almost always one.

training sets. Next, we reorder the data with our toolkit and again create new data sets by removing data with the same proportions as previously.

We then apply data provided for the WMT news translation task[6] for validation and testing. In particular, we use newstest2018 as the development set and newstest2019 as our test set for both language directions. The translation models are trained with the OpenNMT toolkit (Klein et al., 2017) using RNN encoders and decoders with LSTM gates. All training sets are tokenized with the tokenizer from the mosesdecoder toolkit (Koehn et al., 2007) and segmented with BPE (Sennrich et al., 2016) using subword-nmt[7] before feeding them to OpenNMT.

3.1 Ranking

Following Vázquez et al. (2019), we first produce an initial filtering of the ParaCrawl corpus. For this, we use the following heuristic filters from the OpusFilter toolbox:

- LengthFilter: The length of the segments have to be between 1 and 100 words.
- LengthRatioFilter: The maximum ratio between the source and target segments has to be below 3.
- LongWordFilter: Exclude segment pair if any word is longer than 40 characters.
- HtmlTagFilter: Exclude segment pairs with any HTML tags.
- CharacterScoreFilter: All alphabetic characters have to be in Latin script.

The initial filtering removed only 8,055 (0.4%) of the Finnish to English segment pairs, probably because similar filters are already applied in bicleaner when preparing the original data set. Nevertheless, these steps are useful for creating data to train models used in the later filtering methods. First, we train word alignment priors for the model 3 of the eflomal tool[8] (Östling and Tiedemann, 2016) and variable-length character n-gram models for the source and target languages using the VariKN toolkit[9] (Siivola et al., 2007). In addition, we train a background language model that combines the source and target languages of the unfiltered corpus. We interpolate it with the language-specific models with coefficient 0.01 to ensure that we cover all characters that appear in the data.

Next, we take a random subset of 100,000 segment pairs from the corpus for training a logistic regression classifier. To extract features for the logistic regression to be trained on, we use another set of filters from the OpusFilter toolbox:

- CharacterScoreFilter: The proportion of Latin characters among all alphabetic characters
- LanguageIDFilter: Confidence score from the CLD2 language identification library[10] if the correct language is identified, or 0 otherwise
- TerminalPunctuationFilter: The ”term-punct” score from Vázquez et al. (2019)
- NonZeroNumeralsFilter: The ”non-zero” score from Vázquez et al. (2019)
- CrossEntropyFilter: Word-based cross-entropies of the source and target sentences from the respective character n-gram models
- WordAlignFilter: Unnormalized source-to-

[6] http://www.statmt.org/wmt19/translation-task.html
[7] https://github.com/rsennrich/subword-nmt
[8] https://github.com/robertostling/eflomal
[9] https://github.com/vsiivola/variKN
[10] https://github.com/CLD2Owners/cld2

target and target-to-source alignment probabilities obtained by eflomal

Figure 1a shows histograms of the scores over the 100,000 training segments pairs in the data, produced by the `opusfilter-scores` tool. The distribution of the cross-entropy values is quite unimodal, indicating that such a score alone does not make a clear division of the segment pairs as clean or noisy. Language identification scores are mostly close to one, but zero for a small fraction of the segments, indicating that they contain incorrect languages. Also, non-zero numerals and terminal punctuation scores show that a small number of samples look problematic. Word alignment scores have an interesting close-to-bimodal distribution. Smaller values indicate better alignment, so the lower peak is for more problematic segment pairs.

The correlations of the scores over the training data are illustrated in Figure 1b. As excepted, the same scores for source and target segments correlate slightly for all scores and highly for the cross-entropy and alignment scores. Also, non-zero numerals and terminal punctuation filters correlate slightly, indicating segment pairs that have both different punctuation marks and numbers, thus likely to be poor translations. Finally, cross-entropy scores for the source language (Finnish) have a moderate correlation with the alignment scores. As it is likely that the English side has mostly been the original text, problems in the fluency of the translation seem to also indicate issues in its adequacy.

3.2 Results

In this section, we compare the results of models trained with data in the original (bicleaner) order and in the order of our classifier using the different data splits described above. We also test the ROC AUC model for which we created a small development set of 200 randomly selected segment pairs that have manually been annotated as noisy or clean (100 examples each). A pair was annotated noisy only in the case of serious problems; sentences with single translation errors or relatively poor fluency were still considered clean.

Figure 2 provides an overview of the results for Finnish to English. We can see that our filtering method is very effective. Removing noisy data according to the ranking produced by our tool improves the BLEU score compared to the model that applies the whole ParaCrawl data. In contrast, removing data based on the original ParaCrawl order

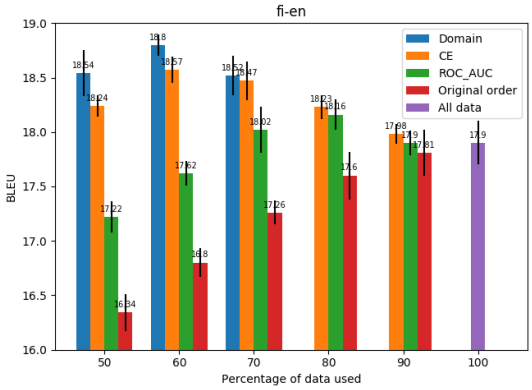


Figure 2: BLEU scores for Finnish-English translation models trained with data that is pruned based on different ranking orders. The reported BLEU values show the mean of six translation models. The 100-mark bar shows the score when using the whole ParaCrawl corpus for training.

degrades the BLEU score at all cutoff points. When using cross-entropy based sorting of the data, cutting off 40% of the lowest scoring training pairs increased BLEU by 0.67 points when compared to using the full training set. If more than 40% of the data is removed, the BLEU score starts to decrease. Surprisingly, ROC AUC based sorting, which requires a manually annotated development set, produces worse results than cross-entropy. ROC AUC reaches a maximum gain of 0.26 BLEU points over using the whole data set when 20% of the data is truncated from the noisy end.

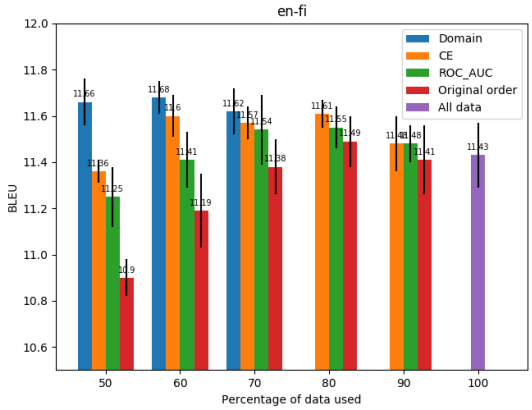


Figure 3: BLEU scores for English-Finnish translation models.

English to Finnish translations show similar results, as illustrated in figure 3, although the BLEU scores are overall lower as it is common in systems translating into morphologically rich languages. Again, cross-entropy based models perform better than ROC AUC based ones: at 80% cutoff cross-

entropy model has 0.18 point and ROC AUC model has 0.12 point improvement over using the whole data. The increases in scores are more modest in English to Finnish translations than in Finnish to English translations.

As seen in Table 1, the cross-entropy based logistic regression model sets the weights of the cross-entropy language model filters and the word alignment filters very close to zero, while setting stronger weights for all other filters compared to the ROC AUC model. Detecting the correct language and having similar numerals in both sides of the sentence pairs seem to be the most important factors for the cleaning task, as their corresponding filters have by far the highest weights.

	CE	ROC AUC
Intercept	-4.63	-4.73
CharacterScoreFilter.src	1.75	0.77
CharacterScoreFilter.tgt	1.22	0.65
CrossEntropyFilter.src	-0.12	0.40
CrossEntropyFilter.tgt	-0.01	0.83
LanguageIDFilter.cld2.src	31.33	11.30
LanguageIDFilter.cld2.tgt	8.39	6.57
NonZeroNumeralsFilter	14.36	13.03
TerminalPunctuationFilter	2.57	0.82
WordAlignFilter.src	-0.15	0.53
WordAlignFilter.tgt	-0.04	0.87

Table 1: Logistic regression weights for models chosen with cross-entropy and ROC AUC for each filter score in the Finnish-English experiment. Positive weight is for the pairs that are predicted as clean.

3.3 OpusFilter for Domain Adaptation

Besides of generally cleaning noisy training data, OpusFilter can also be used to select training data that is similar and appropriate for translation tasks in specific domains. To demonstrate this, we conduct the following domain adaptation experiment.

We use, again, newstest2019 for testing and newstest2018 as development data. To adapt to the news domain, we now take in-domain data from previous years of the news translation task concatenating test sets from 2015, 2016 and 2017 for both Finnish and English. In total, this gives us 7372 sentence pairs that we apply to train n-gram language models for the news domain for both languages using the OpusFilter's **train_ngram** feature.

In Finnish to English translations, the best BLEU score is achieved using 60% of the full training data. To see whether we can reach a higher score by removing training examples that do not fit the news domain, we first select 70% of the cleanest ParaCrawl data based on the order from our cross-entropy optimized classifier. Next, we use our previously trained news domain language models to assign a new score with CrossEntropyFilters for each sentence in both languages in our 70% data. We sort the data based on the language model scores and remove data from the noisy end to create 60% and 50% data sets that reflect the additional domain adaptation. Note that these percentage cutoff points refer to proportions from the full ParaCrawl data set, so the absolute number of sentence pairs is the same as in the other data sets used in the previous experiments. Finally, we apply those news-domain-adapted data sets to train translation models in the same way as before.[11]

The results are included in Figures 2 and 3. In all cases, the domain filtering leads to an improvement compared to the corresponding noise-filtered model. At the 70% mark, the results are very similar as the training sets are essentially the same. The Finnish to English model improves the score by 0.23 BLEU points over the noise-filtered model at the 60% mark. The English to Finnish model produces similar results but with lower scores. Those results demonstrate the effectiveness of OpusFilter to also perform data selection for domain adaptation without further annotation and additional components.

4 Conclusions and Future Work

This paper introduces OpusFilter, a modular tool for parallel data selection and ranking. OpusFilter can easily be configured to work with OPUS data and various filters to train effective classifiers in order to rank bitext segments. We demonstrate its use in a Finnish-English translation task based on the noisy ParaCrawl data used for training. The classifiers can be trained without human annotation, and the automatic model selection methods implemented in the toolbox lead to a similar performance compared to classifiers based on small manually labeled validation data. OpusFilter is open source and distributed with a permissive license to make it widely applicable. In future work, we would like to extend the toolbox with additional filters and classification options. One option could be the inclusion of sentence embedding based filtering (Guo et al., 2018). Additionally, we would like to explore OpusFilter's use in different scenarios and for other language pairs. Especially interesting

[11] The WMT testsets are not included in training the models.

would be the application in low-resource settings and various levels of noise in the original data. Furthermore, the use for domain adaptation and data selection should be further explored.

Acknowledgments

This work is part of the FoTran project, funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement № 771113), as well as the MeMAD project, funded by the European Union's Horizon 2020 Research and Innovation Programme (grant agreement № 780069).

References

Miquel Esplà-Gomis, Mikel L Forcada, Gema Ramírez-Sánchez, and Hieu Hoang. 2019. Paracrawl: Web-scale parallel corpora for the languages of the eu. In *Proceedings of Machine Translation Summit XVII Volume 2: Translator, Project and User Tracks*, pages 118–119.

Mandy Guo, Qinlan Shen, Yinfei Yang, Heming Ge, Daniel Cer, Gustavo Hernández Ábrego, Keith Stevens, Noah Constant, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. 2018. Effective parallel corpus mining using bilingual sentence embeddings. *CoRR*, abs/1807.11906.

Huda Khayrallah and Philipp Koehn. 2018. On the impact of various types of noise on neural machine translation. In *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, pages 74–83, Melbourne, Australia. Association for Computational Linguistics.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. 2017. Opennmt: Open-source toolkit for neural machine translation. In *Proc. ACL*.

Philipp Koehn, Francisco Guzmán, Vishrav Chaudhary, and Juan Pino. 2019. Findings of the wmt 2019 shared task on parallel corpus filtering for low-resource conditions. In *Proceedings of the Fourth Conference on Machine Translation (Volume 3: Shared Task Papers, Day 2)*, pages 56–74, Florence, Italy. Association for Computational Linguistics.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 177–180, Prague, Czech Republic. Association for Computational Linguistics.

Philipp Koehn, Huda Khayrallah, Kenneth Heafield, and Mikel L. Forcada. 2018. Findings of the wmt 2018 shared task on parallel corpus filtering. In *Proceedings of the Third Conference on Machine Translation, Volume 2: Shared Task Papers*, pages 739–752, Belgium, Brussels. Association for Computational Linguistics.

Robert Östling and Jörg Tiedemann. 2016. Efficient word alignment with Markov Chain Monte Carlo. *Prague Bulletin of Mathematical Linguistics*, 106:125–146.

Víctor M. Sánchez-Cartagena, Marta Bañón, Sergio Ortiz-Rojas, and Gema Ramírez-Sánchez. 2018. Prompsit's submission to wmt 2018 parallel corpus filtering shared task. In *Proceedings of the Third Conference on Machine Translation, Volume 2: Shared Task Papers*, Brussels, Belgium. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Vesa Siivola, Teemu Hirsimäki, and Sami Virpioja. 2007. On growing and pruning Kneser-Ney smoothed n-gram models. *IEEE Transactions on Audio, Speech and Language Processing*, 15(5):1617–1624.

Jörg Tiedemann. 2012. Parallel data, tools and interfaces in OPUS. In *Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey. European Language Resources Association (ELRA).

Raúl Vázquez, Umut Sulubacak, and Jörg Tiedemann. 2019. The university of Helsinki submission to the WMT19 parallel corpus filtering task. In *Proceedings of the Fourth Conference on Machine Translation (Volume 3: Shared Task Papers, Day 2)*, pages 294–300, Florence, Italy. Association for Computational Linguistics.

Hainan Xu and Philipp Koehn. 2017. Zipporah: a fast and scalable data cleaning system for noisy web-crawled parallel corpora. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2945–2950, Copenhagen, Denmark. Association for Computational Linguistics.